Assessing the Performance of Large MPI Applications Instances Through Time-Independent Traces

F. Desprez[1] **G. Markomanolis**[1] F. Suter[2]

[1]INRIA, LIP, Avalon, ENS de Lyon [2]IN2P3 Computing Center, CNRS, IN2P3, Avalon

Grid'5000 Winter School
Nantes, 2012

PICS 5473

ANR 11 INFRA 13

Context and motivation

Context

- Simulation: popular approach to get objective performance indicators
- May help the dimensioning of compute clusters
- Two complementary approaches
  - On-line: execute the application with some simulated parts
  - Off-line: replay an execution trace

Motivation

- Off-line simulation usually based on timed traces
  - Link trace to acquisition
- Proposition: get rid off the timestamps
  - Decouple acquisition from actual replay
- Target: regular data-independent MPI applications

Outline

Time-Independent Trace Format

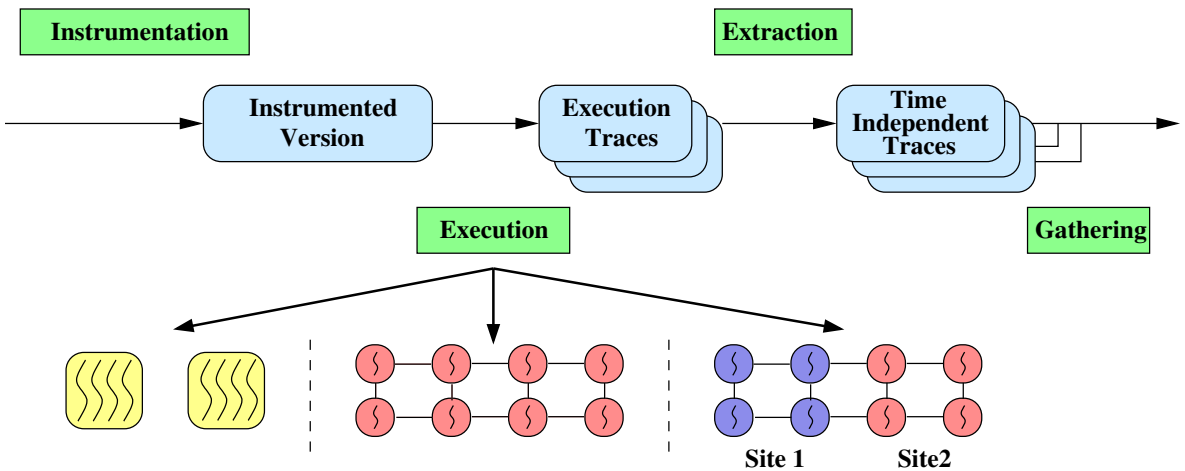
```
for (i=0; i<4; i++){
 if (myId == 0){
  /* Compute 1M instructions */
  MPI_Send(1MB,..., (myId+1));
  MPI_Recv(...);
 } else {
  MPI_Recv(...);
  /* Compute 1M instructions */
  MPI_Send(1MB,..., (myId+1)% nproc);
 }
}
```

```
0 compute 1e6
0 send 1 1e6
0 recv 3 1e6

1 recv 0 1e6
1 compute 1e6
1 send 2 1e6

2 recv 1 1e6
2 compute 1e6
2 send 3 1e6

3 recv 2 1e6
3 compute 1e6
3 send 0 1e6
```

- List of actions performed by each process
- Each action is constituted by:
  - id of the process
  - type, e.g., computation or communication
  - volume in instructions or bytes
  - action specific parameters

Trace Acquisition Process

```
for (i=0; i<4; i++){
 if (myId == 0){
  /* Compute 1Mflop */
  MPI_Send(..., (myId+1));
  MPI_Recv(...);
 } else {
  MPI_Recv(...);
  /* Compute 1Mflop */
  MPI_Send(...,
         (myId+1)% nproc);
 }
}
```



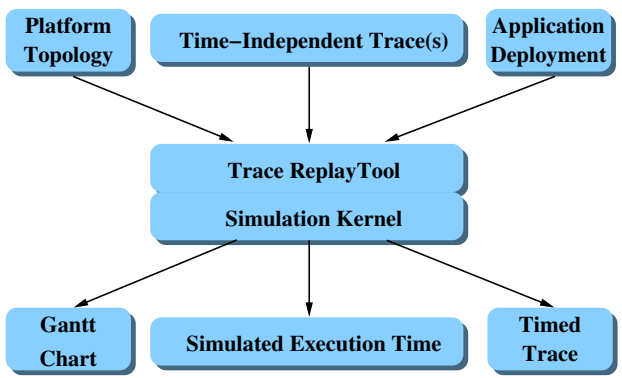

```
0 compute 1e6
0 send 1 1e6
0 recv 3

1 recv 0
1 compute 1e6
1 send 2 1e6

2 recv 1
2 compute 1e6
2 send 3 1e6

3 recv 2
3 compute 1e6
3 send 0 1e6
```

Trace Replay with SimGrid

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="3">
  <cluster id="cluster" prefix="c-"
           suffix=".me" radical="0-3"
           power="1E9" bw="1.25E8" lat="15E-6"
           bb_bw="1.25E9" bblat="15e-6"/>
</platform>
```

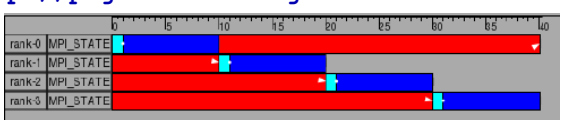
```
0 compute 1e6
0 send 1 1e6
0 recv 3 1e6

1 recv 0 1e6
1 compute 1e6
1 send 2 1e6

2 recv 1 1e6
2 compute 1e6
2 send 3 1e6

3 recv 2 1e6
3 compute 1e6
3 send 0 1e6
```

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="3">
  <process host="c-0.me" function="0"/>
  <process host="c-1.me" function="1"/>
  <process host="c-2.me" function="2"/>
  <process host="c-3.me" function="3"/>
</platform>
```



http://simgrid.gforge.inria.fr

http://paje.sourceforge.net




```
Simulated time:
  0.0401133
```

```
[0.001000] 0 compute 1e6 0.01000
[0.010028] 0 send 1 1e6 0.009028
[0.040113] 0 recv 3 1e6 0.030085

[0.010028] 1 recv 0 1e6 0.010028
...
```

Main issue: Calibration

Platform file : instructions per second of the CPU, latency, bandwidth

- These values are needed for accurate performance predictions

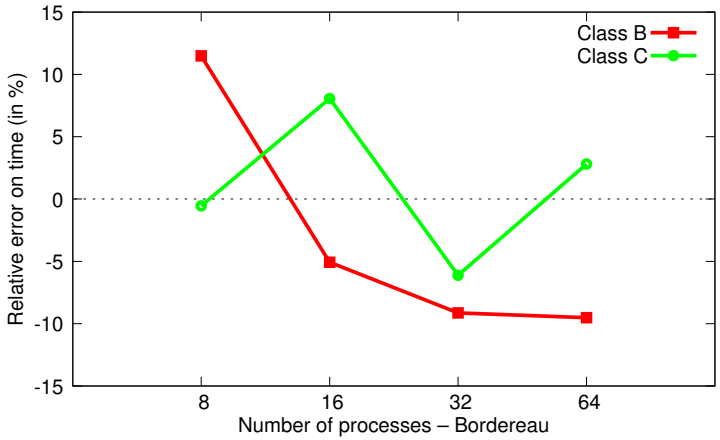
Computation

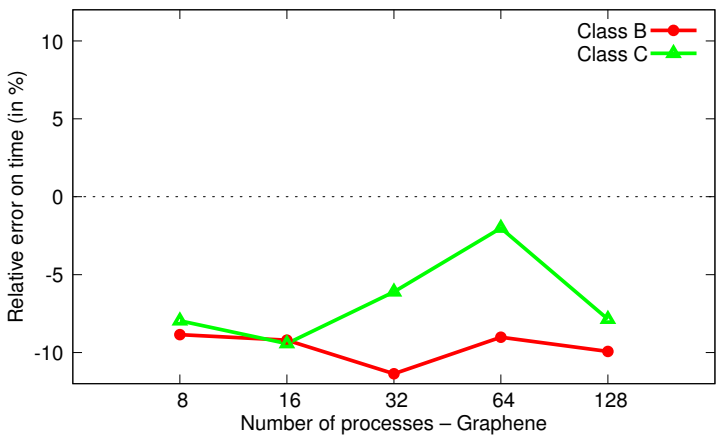
- Execute a small instrumented instance of the target application
- Compute the instruction rate for every event
- Compute a weighted average of the instruction rates for each process
- Compute the average instructions rate for all the process set
- Compute an average over these five runs

Communication

- Bandwidth: We use the nominal value of the links
- SkaMPI for measuring the latency
- Piece-wise linear model used by SIMGrid dedicated to MPI communications:
  - Latency and bandwidth correction factors

Simulation Accuracy

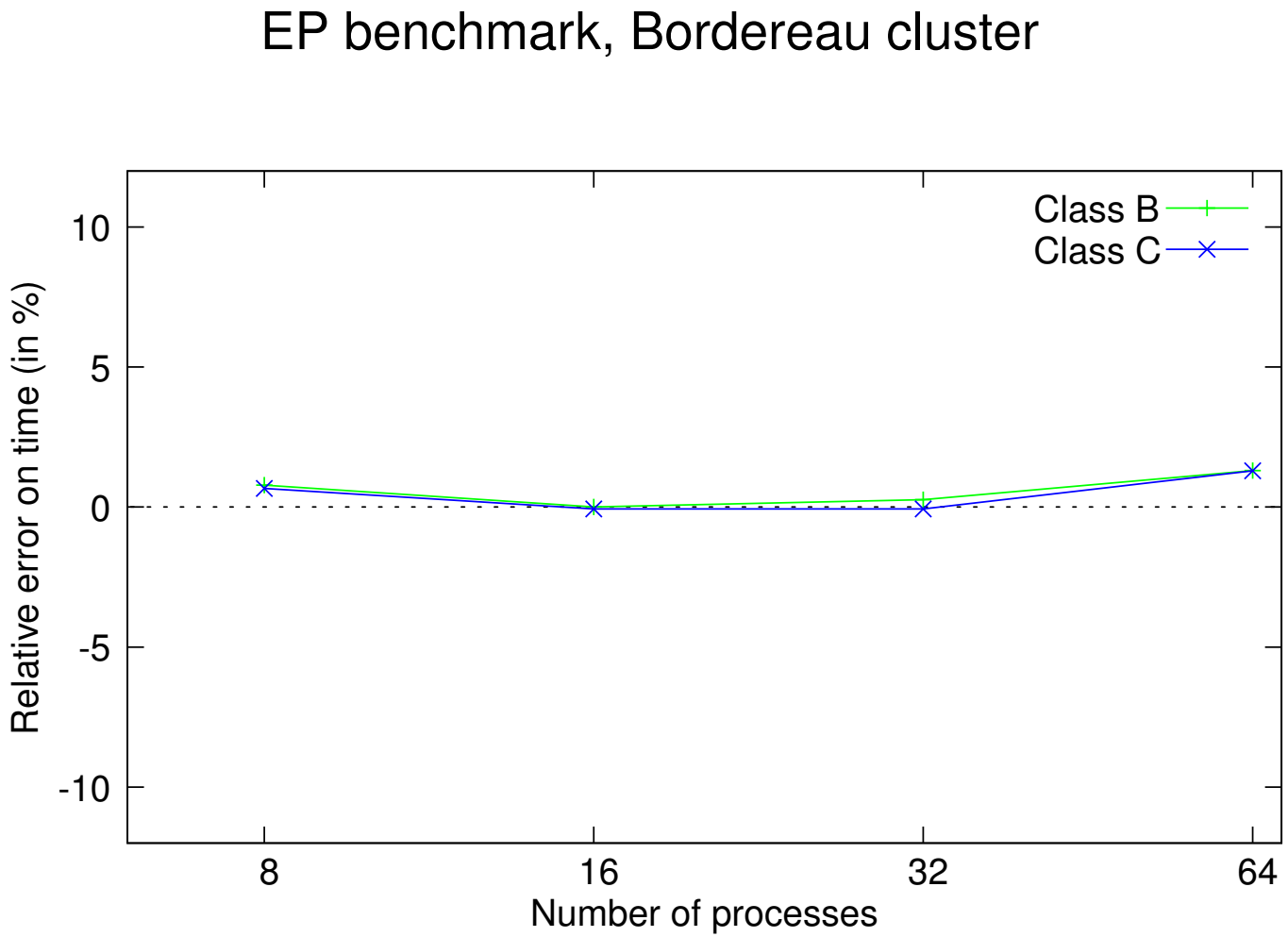




- Greater accuracy
  - Roughly in a [-10%; +10%] interval
  - Impact of all modifications
- Decreasing trend for Class B
  - Small messages are underestimated
    - Time to copy data in memory
- On Graphene cluster
  - Results even more stable
    - in a [-11.4%; -2%] interval
  - General underestimation
    - Due to small messages

- General conclusions
  - Simple modifications $\Rightarrow$ Great improvement! (10% more accurate)
    $\Rightarrow$ Performance prediction is possible now!

Simulation Accuracy



- The worst relative error for EP is 1.3%

Outline

Trusting the Hardware

- Are the results correct?
- Is there any faulty node?

Script for identifying faulty nodes (at least some)

- Give a machine file as input
  - Extract one machine file per cluster
  - Execute the LU-B benchmark (example for 50 nodes):
    - Create one machine file with the the first 32 nodes
    - Create a second one machine file with the last 32 nodes
    - Create a third machine file with the 18 last nodes + 14 first nodes
  - Compare the relative difference of the timings
    - Adaptive value of what we mean “error”
  - If there is any machine file with a faulty node, repeat the procedure on this specific machine file
  - If some faulty nodes are found, remove them from the initial machine file
- More ideas to be implemented

Trusting the Hardware

Real case, Sophia site

Executing LU-B-16:
First machine file: 22.97 seconds
Second machine file: 49.37 seconds
Identified faulty nodes: suno-5 or suno-45

Real case, Bordeaux site

Executing LU-B-32:
First machine file: 20.41 seconds
Second machine file: 23.67 seconds
Third machine file: 24.97 seconds
Identified faulty node: bordereau-72

Trusting the Hardware

- Can we identify easily nodes with faulty memory?
- MPI application and node with faulty memory
- Memtester?
  - Needs more than 1.5 hour for testing 48GB of memory

Script for identifying nodes with faulty memory (at least some)

- Choose one node of each cluster
- Calibrate how many MPI processes of a hello world application can be executed on a node
- TakTuk + 1 mpirun per node, per cluster with almost full memory
- If the node prints its hostname, then it works, ortherwise is crashed because of faulty memory or non homogeneous characteristics

Identified node: Stremi-19 (error on 17-18th memory dimm)

Trusting the Hardware

- Known issue according to other users/admins
- Grid'5000 API does not always give correct results to the requests
- According to Grid'5000 API the node bordereau-18 has 4 GB of memory while it has 2 GB.
- Using TakTuk in order to extract information about the nodes is more safe
- This is important as we reconstruct the machine file according to the available memory of a node

Non homogeneous nodes: Bordereau-18, Genepi-34

Outline

Folding mode

- Choose a node with a lot of memory (*stremi* cluster, 48GB memory, using 24 cores)
- Testing EP benchmark, classes B,C,D for 2048 MPI processes

Class	#Processes	Time in seconds			
		TAU	Scalasca	Score-P	Manual Instrumentation
B	2048	X	134.2	8000	50
C	2048	X	169	X	76.25
D	2048	X	650	X	598

#Processes	Maximum memory in GB			
	TAU	Scalasca	Score-P	Manual Instrumentation
2048	X	20.1	23.6	16

- We can acquire the traces for 2048 nodes just from one node
- TAU needs much more memory than the other tools
- Score-P needs too much time for synchronization issues and unification of the traces

Folding mode

- Testing LU benchmark, instances B-256, C-1024, D-256

Class	#Processes	Time in seconds			
		TAU	Scalasca	Score-P	Modified MPE
B	256	170	124.2	110.5	112
C	1024	X	976	1637.5	775
D	256	X	3312	X	2843

Class	#Processes	Maximum memory in GB			
		TAU	Scalasca	Score-P	Modified MPE
B	256	21.4	2.8	5.2	1.65
C	1024	X	12.9	29.3	7.95
D	256	X	16.9	X	15.4

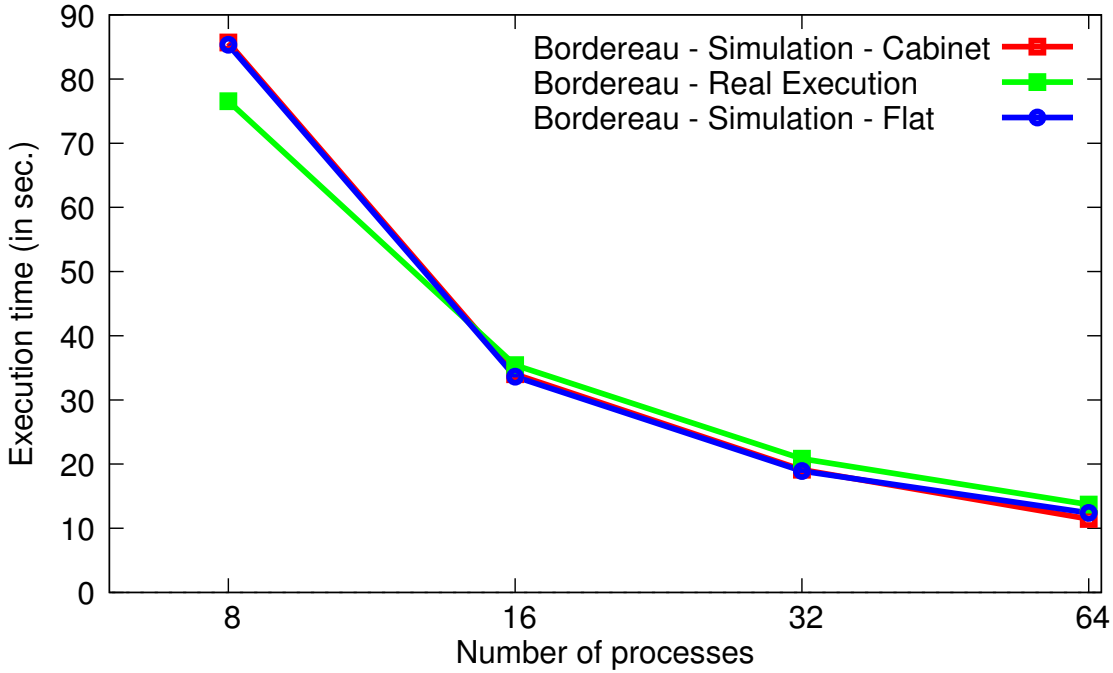
- What is the modified MPE?

Modified MPE

- Why a “new” tool?
  - The most of the well known tool do not consider the folding mode as a use case
  - Some of the issues:
    - A lot of memory needed
    - Slow to save files when a core handles a lot of traces
  - Is it a solution? (work under progress)
    - The MPE tool produces actions according to the TIT format
    - Added support for some PMPI calls, like PMPI_Irecv with ANY_SOURCE
    - Added PAPI support for the compute actions
    - The traces are already in TIT format
  - TODO:
    - Selective instrumentation
    - Validate and solve issues with the value of the total instructions

Scenario I

- What if the *bordereau* cluster had the same topology with *graphene*?
- LU benchmark, class B



- Bordereau: AMD Opteron 2218, 2.6 GHz, 1MB L2 cache per core
- Graphene: Intel Xeon x3440, 2.53 GHz, 2MB L2 cache per core
- Cabinet1: 1-39, Cabinet2: 40-74 processors
- Bordereau cluster behaves better with the original topology till 32 nodes
- For the B-64 instance, the size of the messages belong to the interval of the piece-wise model that graphene has higher bandwidth and only 1% of the messages are bigger than 64 KB

Scenario II

- We want to use as many resources possible and acquire the biggest instance for the LU benchmark with the TAU tool
- Reservation: 778 nodes, 9 sites (except Lille), 18 clusters
- Create a machine file according to the nodes' memory size
- Class D, 4096 MPI processes

Results

Execution time: 1564 seconds
TAU traces size: 420 GB
Converting to TIT traces: 42 seconds (using 778 hard disks)
TIT traces size: 209GB
Gathering time: 245 seconds (compressed files)

Scenario III

- We want to execute LU benchmark, class E
- Reservation: 778 nodes, 9 sites (except Lille), 18 clusters
- Create a machine file according to the nodes' memory size
- 16384 MPI processes
- TAU fails, SCORE-P never finishes (cacelled after 34260 seconds), Scalasca issue with converting to TIT traces
- Using modified MPE

Results

Execution time: 4114 seconds
Trace sizes: 1453 GB
Gathering time: 969 seconds (compressed files)

Scenario IV

- Execute a big instance of the EP benchmark
- Reservation: 778 nodes, 9 sites (except Lille), 18 clusters
- Class D, 32768 MPI processes
- Manual instrumentation

Results

Execution time: 123.38 seconds
Trace sizes: 128 MB
Gathering time: 27.2 seconds (compressed files)

- No luck to execute the instance D-65536, although we had reserved nodes with the needed memory (9 TB)

Gathering Traces

- Do we need another one tool for gathering files?
- 778 nodes, on 9 sites
- EP benchmark

Class	#Processes	Time in seconds		MB
		Kaget	Trace Gather	
B	1024	7.7	20.5	4.1
B	4096	8	21.89	27
B	8192	12.1	22.64	36
B	16384	12.3	26.57	68

- LU benchmark

Class	#Processes	Time in seconds			MB
		Kaget	Trace Gather	Compressed	
C	256	127	77	12.2	3900
C	512	266	206	25.81	8100
C	1024	510	376	38	17000

- Kaget is efficient for small files but not for big ones (considering that we did not miss any advanced option)

Conclusion and Future Work

- Conclusion
  - We used a lot of nodes in order to stress our tools
  - We identified many issues
  - Trust a machine before you use it
  - A benchmark suite is needed for being able for a user to check the hardware any moment
  - Stability of the nodes
  - We can test many “what-if” scenarios
- Future Work
  - Handle memory copy time when sending small messages
  - Automate the calibration procedure
  - Assess performance of real MPI application
  - Check how to improve the discovery of faulty nodes

Thank you!
Questions?